**Query format:**

*? - table\_to\_sop( INPUT\_VARIABLES , LIST\_OF\_LISTS, OUTPUT\_VARIABLE ).*

*?- table\_to\_pos( INPUT\_VARIABLES , LIST\_OF\_LISTS, OUTPUT\_VARIABLE ).*

**The input consists of:**

 A list of variable names as the first argument, a list of lists (truth table) as the second argument, and a variable in which the result is returned as the final argument. Note that, the list of lists has been defined with the **output first i.e. the first element of each component list of the list of lists corresponds to the output of truth table.**

**Examples:**

?- table\_to\_sop( [ "A" ,"B" , "C"] , [[1,1,1,0],[1,1,0,0],[0,1,1,1],[1,1,1,1]], Z ).

OUTPUT:

Tree Representation: *or*([*and*(["A", "B", "!C"]), *and*(["A", "!B", "!C"]), *and*(["A", "B", "C"])])

**Z** = [["A", "B", "!C"], ["A", "!B", "!C"], ["A", "B", "C"]]

? - table\_to\_pos( [ "A" ,"B" , "C"] , [[1,1,1,0],[1,1,0,0],[0,1,1,1],[1,1,1,1]], Z ).

OUTPUT:

Tree Representation: *or*([["!A", "!B", "!C"]])

**Z** = [["!A", "!B", "!C"]]

? - table\_to\_sop( [ "L" ,"M" , "N"] , [[0,1,1,0],[0,1,0,0],[0,1,1,1]], Z ).

OUTPUT:

Tree Representation: []

**Z** = []

? - table\_to\_sop( [ "A" ,"B" , "C"] , [[1,2,1,0],[1,1,0,0],[0,1,1,1],[1,1,1,1]], Z ).

OUTPUT: "invalid input...please check" **false**

**Important predicates:-**

1. **table\_to\_sop:** takes the truth table as input(in form of list of lists), along with a list of variables to be used and:

* Represents it in form a 2-level circuit using AND and OR gates with Sum of Products implementation.
* gives Boolean expression

1. **table\_to\_pos:** (inputs same as above)

* Represents it in form a 2-level circuit using AND and OR gates with Product of Sums implementation.
* gives out Boolean expression

1. **check\_it\_sop([H|T3] , Variables2 ,RT ,Final\_list) :**

Uses recursion to take list ,one list at a time from the input(list of lists) and see if it’s corresponding output in the truth table is 1 or not.

[H|T3] – input (list of lists)

. If there is any other truth table output apart from 1 and 0, the program shows an “invalid input” message and terminates.

Variables2 – list of variables used as inputs for truth table.

RT – used as an accumulator.

Final\_list - output

1. **check\_it\_final( Equal , Equal)**: Makes Final\_list = equated to Reversed list-of-list (called from the 3rd predicate)
2. **term3\_sop([H2|T2], [Hv|Tv] , LOut ,RT):**

It gives the individual terms of Boolean expression (output) in form of a list.

The if clause checks whether a given input in the truth table is 1 or 0. If it’s 1, the variable is simply appended in the output list, and if it’s 0, it’s compliment (taken using String\_concat) is appended to the output list. If there is any other input apart from 1 and 0, the program shows an “invalid input” message and terminates. The output is an implementation of sum of products.

1. **term3\_pos([H2|T2], [Hv|Tv] , LOut ,RT):**

Its function is same as the fifth predicate, the difference being in the implementation of product of sums instead of sum of products.

1. The predicates **accRev([H|T] , A ,R)** and **rev(L , R)** are used to reverse the given list using an accumulator.
2. **print\_sop( List , Output):**

This function gives the final output in the sum of products form (thus, implementing the 2-level circuit using AND and OR gates).

For example, if the given Boolean function is:

*F = A’BC + ABC’*

Then it’s represented as: *or(and(!A,B,C), and(A,B,!C))*

***NOTE: print\_pos(List, Output)*** *also does the same, the difference being in the the “product of sums” implementation.*

1. **convert\_to\_AND(Z,and(Z))** and **convert\_to\_OR(L , or(L))** are the predicates called from print\_sop and print\_pos. This indicates the use of functors ‘and’ and ‘or’.
2. **step1\_pos([H|T] , L ,W)** and **step1\_sop([H|T] , L ,W)** are also predicates called from print\_pos and print\_sop respectively (Help in recursion).